

JPA

Java Persistence API

Técnicas de Programação
FACIN-PUCRS
Prof. Leandro Pompermaier
2008/02

Introdução

- Em aplicativos de tamanho médio a grande vale a pena organizar as classes por assuntos ou “camadas”.
- Vantagem: alterações em uma camada não implica em rompimento com as demais.
- Camadas típicas (horizontais):
 - Apresentação: interface com o usuário (ou outros sistemas).
 - Lógica de negócio: implementa o problema do domínio da aplicação.
 - Persistência: responsável pela memória dos dados e sua recuperação.

Introdução

- Alternativas para a camada de persistência em sistemas O.O.:
 - Arquivos clássicos;
 - Serialização;
 - SQL/JDBC;
 - Bancos de dados O.O.
 - Tecnologia ainda em desenvolvimento
 - Incompatíveis com os sistemas legados da empresa

Introdução

- Problemas para se trabalhar no modelo E/R
 - Trabalha-se no modelo dos dados
 - Forte acoplamento c/os elementos do modelo E/R: tabelas
 - Característico de programação procedural
 - Objetos apenas para “uso em memória”
 - Como navegar pelo diagrama de objetos?
 - `Obj1.metodo1().getDado2().getDado3();`
 - Como mapear hierarquias de herança?
 - Como tratar a questão das chaves primárias?
 - Qual o ciclo de vida de um objeto persistente?
 - Como representar as associações entre objetos?

Introdução

- Mapeamento Objeto Relacional (ORM)
 - Camada que mapeia os conceitos de O.O. para E/R e vice-versa.
 - Vantagens:
 - Produtividade
 - Trabalha-se sempre no mesmo paradigma (O.O.)
 - Manutenção
 - Baixo acoplamento com o modelo de dados
 - Desempenho???
 - Porque não fazer tudo em assembler?

Introdução

- Java Persistence API: um framework que pode ser usado para implementar a camada de persistência usando mapeamento objeto-relacional.
- Se apresenta na forma de uma biblioteca de classes e arquivos de configuração.
- Permite que o desenvolvedor trabalhe com o modelo de objetos deixando para a JPA a tarefa de persistir os mesmos no modelo relacional.

Introdução

- Existem algumas implementações
 - Toplink
 - Hibernate
 - GlassFish
- Todos implementam a solução ORM de acordo com a especificação JPA
- Têm o formato de arquivos *.jar que adicionados ao classPath da aplicação tornam-se disponíveis

JPA

- Ferramenta de mapeamento objeto relacional:
 - Usa metadados para orientar o mapeamento entre modelos.
- JDK 5.0 e 6.0
 - Suporta anotações.
 - Pré-processamento durante a compilação
- Anotações:
 - Embutidas nos bytecodes e lidas em tempo de execução.
 - No caso do JPA são lidas na inicialização do sistema

JPA: Conceitos Básicos

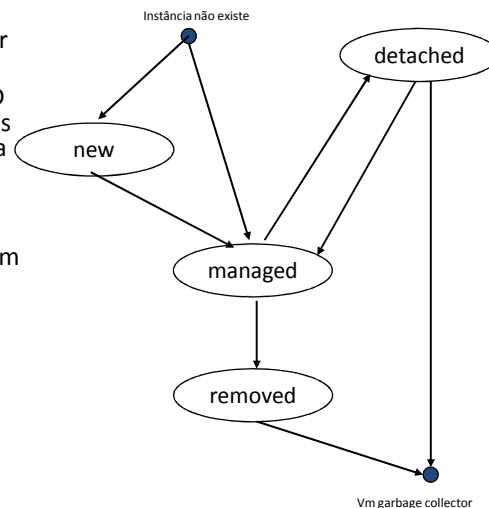
- Objetos persistentes = entidades
 - Implementados como POJOs
 - Podem explorar herança e classes abstratas
- Entidades
 - Possuem um identificador (chave primária)
 - Possuem um estado
 - Ciclo de vida independente do tempo de vida da aplicação (podem ser compartilhadas por diferentes aplicações)

JPA: Conceitos Básicos

- Persistence context: associação entre as instâncias e o BD
- Entity Manager: responsável pela execução das operações básicas (CRUD + consulta). Obtido a partir do “persistence context”
- Persistence provider: é que implementa as interfaces definidas pela JPA (exs: Oracle TopLink Essentials, Hibernate Entity Manager, BEA Kodo, Apache OpenJPA)

JPA: Ciclo de vida de objeto

- Na situação **new** o objeto não está associado ao EntityManager
- Na situação **managed** o EntityManager garante que o BD será atualizado com o estado dos atributos persistentes ao final da transação.
- Na situação **detached** o objeto não está ligado a um PersistentContent, mas possui um registro correspondente no BD
- Na situação **removed** o objeto possui registro correspondente no BD mas foi marcado para remoção ao final da transação.



JPA: Configuração de Ambiente

- Instalar um SGBD
 - Nos exemplos de aula será usado o HSQLDB, porém, aconselha-se o uso de um SGBD mais robusto
 - No caso do HSQLDB baixar e descompactar o arquivo de instalação. O arquivo hsqldb.jar, presente no diretório lib, deverá fazer parte do projeto da aplicação (projeto eclipse).
 - Download: <http://www.hsqldb.org/>
- Instalar uma implementação da JPA
 - O TopLink é a implementação da JPA usada no projeto GlassFish.
 - Download: <http://www.oracle.com/technology/products/ias/toplink/jpa/download.htm>
 - Após o download executar o comando: "java -jar glassfish-persistence-installer-v2-b15.jar. Isso extrai, entre outros, o arquivo "toplink-essentials.jar" que deve ser adicionado ao projeto eclipse.

Arquivos de configuração

- JPA → um arquivo de configuração
 - Especifica o driver do banco de dados
 - Especifica quais as entidades serão mapeadas
 - Nome: META-INF/persistence.xml
 - Deve fazer parte do projeto
 - Atenção: o diretório META-INF deve estar localizado no mesmo diretório da execução (no Eclipse, pode ser inicialmente colocado no “src”. O Eclipse duplica de acordo)

Exemplo: persistence.xml (p1)

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="empregados">
    <provider>
      oracle.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider
    </provider>

    <class>Departamento</class>
    <class>Empregado</class>
```

> Continua <

- Nome da base de dados
- Nome da implementação da JPA
- Relação de classes a serem persistidas

Exemplo: persistence.xml (p2)

...

```
<properties>
  <property name="toplink.jdbc.driver" value="org.hsqldb.jdbcDriver"/>
  <property name="toplink.jdbc.url" value="jdbc:hsqldb:mem:."/>
  <property name="toplink.jdbc.user" value="sa" />
  <property name="toplink.jdbc.password" value="" />
  <property name="toplink.ddl-generation" value="create-tables"/>
  <property name="toplink.logging.level" value="INFO"/>
  <property name="toplink.platform.class.name"
    value="oracle.toplink.essentials.platform.database.HSQLPlatform"/>
</properties>
</persistence-unit>
</persistence>
```

- Nome do driver do BD
- URL de acesso ao BD
- User/pass de acesso ao BD
- Cria tabelas novas a cada execução

JPA Annotations

JPA Annotations

- **@Entity**
 - Define o POJO, indicando o conceito a ser persistido

```
@Entity
public class Aluno {
    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations

- Os atributos são persistida automaticamente, sem a necessidade de anotações.
 - O nome do atributo corresponde ao nome de campo que a representa na tabela (caso queira alterar este nome as anotações serão necessárias)

```
@Entity
public class Aluno {

    private int matricula;

    private String nome;

    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations

- **@transiente**
 - Atributo que não será persistido

```
@Entity
public class Aluno {

    private int matricula;

    private String nome;

    @transient
    private boolean isAtivo = true;
    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations

- **@Column**
 - Utilizado quando desejamos alterar o nome da coluna, ou seja, o atributo não possui o mesmo nome do campo da tabela.

```
@Entity
public class Aluno {

    @Column(name="MatriculaAluno")
    private int matricula;

    @Column(name="NomeAluno")
    private String nome;

    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations

- **@Id**
 - O atributo representa o identificador do objeto

```
@Entity
public class Aluno {
    @Id
    @Column(name="MatriculaAluno")
    private int matricula;

    @Column(name="NomeAluno")
    private String nome;

    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations

- Quer definir a estratégia de geração de Ids?
 - basta utilizar a anotação `@GeneratedValue` e definir a estratégia.

```
@Entity
public class Ramal {

    @Id
    @Column(name="id")
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private int id;

    @Column(name = "numero")
    private int numero;
```

JPA Annotations

- **@Table**

- Permite definir o nome da tabela diferente do nome da classe

```
@Entity
@Table(name="Alunos")
public class Aluno {
    @Id
    @Column(name="MatriculaAluno")
    private int matricula;

    @Column(name="NomeAluno")
    private String nome;
    ...
}
```

Aluno
-matricula: int
-nome: String

JPA Annotations – Relacionamentos

- **@OneToOne**

- Define o relacionamento um-para-um

```
@Entity
@Table(name="Alunos")
public class Aluno {
    @Id
    @Column(name="MatriculaAluno")
    private int matricula;

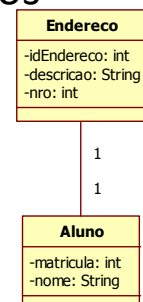
    @Column(name="NomeAluno")
    private String nome;

    @OneToOne(mappedBy="aluno")
    private Endereco endereco;
    ... }
}
```

```
@Entity
@Table(name="Enderecos")
public class Endereco {

    @Id
    private int idEndereco;
    private String descricao;
    private int nro;

    @OneToOne
    @JoinColumn(name="aluno_MatriculaAluno")
    private Aluno aluno;
    ...}
}
```



JPA Annotations – Relacionamentos

- @ManyToOne

```
@Entity
public class Nota {

    @Id
    private int idNota;
    private String descricao;
    private double notaAluno;

    @ManyToOne(cascade=ALL)
    @JoinColumn(name="aluno_MatriculaAluno",
        referencedColumnName = "MatriculaAluno")
    private Aluno aluno;

    ...}

```



JPA Annotations – Relacionamentos

- @OneToMany

```
@Entity
@Table(name="Alunos")
public class Aluno {
    @Id
    @Column(name="MatriculaAluno")
    private int matricula;

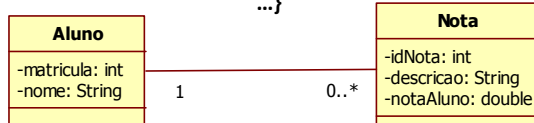
    @Column(name="NomeAluno")
    private String nome;

    @OneToOne(mappedBy="aluno")
    private Endereco endereco;

    @OneToMany(cascade=ALL, mappedBy = "aluno")
    private List<Nota> notas = new ArrayList<Nota>();

    ...}

```



JPA Annotations – Relacionamentos

- @ManyToMany

```
@Entity
@Table(name="Alunos")
public class Aluno {
```



```
@ManyToMany
@JoinTable(name="DisciplinaAluno",
    joinColumns = @JoinColumn(name="Aluno_MatriculaAluno",
        referencedColumnName = "MatriculaAluno"),
    inverseJoinColumns =
        @JoinColumn(name="disciplinas_codigo",
            referencedColumnName = "codigo"))
private List<Disciplina> disciplinas = new ArrayList<Disciplina>();
```

JPA Annotations – Relacionamentos

- @ManyToMany

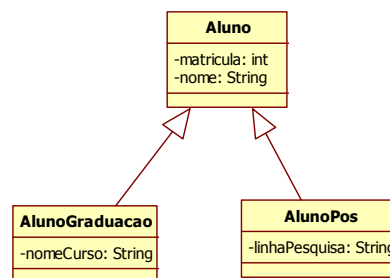


```
@Entity
public class Disciplina {
```

```
@ManyToMany
@JoinTable(name="DisciplinaAluno",
    joinColumns = @JoinColumn(name="Disciplina_codigo",
        referencedColumnName = "codigo"),
    inverseJoinColumns =
        @JoinColumn(name="alunos_MatriculaAluno",
            referencedColumnName = "MatriculaAluno"))
List <Aluno> alunos = new ArrayList<Aluno>();
```

JPA Annotations – Relacionamentos

- Herança
 - Existem várias estratégias de implementação de herança
 - A abordagem estudada usa uma única tabela com um campo discriminador



JPA Annotations – Herança – Classe Pai

```

@Entity
@Table(name="Alunos")
@Inheritance(strategy=SINGLE_TABLE)
@DiscriminatorColumn(name="tipo", discriminatorType = STRING)
public abstract class Aluno {
    @Id
    @Column(name="MatriculaAluno")
    private int matricula;

    @Column(name="tipo", nullable=false)
    private int tipo;
    public abstract String getTipo();
  
```

Observe a definição da estratégia de tabela única

Observe a definição da coluna discriminatória

JPA Annotations – Herança – Classe Filha

```

@Entity
@DiscriminatorColumn(name="tipo", discriminatorType = STRING)
@DiscriminatorValue("GRADUACAO")
public class AlunoGraduacao extends Aluno {

    private String nomeCurso;

    public AlunoGraduacao() {

    }

    @Override
    public String getTipo() {
        return "GRADUACAO";
    }

}

```

Observe o valor discriminante

Observe o indicador de sobrecarga

JPA Annotations – Relacionamentos

```

@Entity
@DiscriminatorColumn(name="tipo", discriminatorType = STRING)
@DiscriminatorValue("POS")
public class AlunoPos extends Aluno {

    private String nomeCurso;

    public AlunoPos() {

    }

    @Override
    public String getTipo() {
        return "POS";
    }

}


```


Obtendo acesso ao EntityManager

```
factory = Persistence.createEntityManagerFactory("locadora");
```

```
manager = factory.createEntityManager();
```

Nome da base de dados definido
no arquivo de configuração



Classe DAO

```
import javax.persistence.*;
import java.util.*;

public class AlunoDAO {
    private static AlunoDAO instance = null;
    private EntityManager manager;

    public static AlunoDAO getInstance(){
        if (instance == null){
            instance = new AlunoDAO();
        }
        return(instance);
    }

    public void setManager(EntityManager mng){ manager = mng; }

    public void popula(){ ... }

    public Aluno recupera(int nro){ ... }

    public void remove(int nro){ ... }

    public void atualiza(Aluno aluno){ ... }

    public List<Aluno> getAllAlunos(){ ... }
}
```

- Esta classe concentra todos os métodos de acesso a base de dados.
- Note que os parâmetros das operações são apenas instâncias de *Aluno* ou o campo usado para a pesquisa

DAO: persistindo uma entidade

```
public void atualiza(Aluno aluno) {
    try {
        manager.persist(aluno);
    } catch (RuntimeException e) {
        e.printStackTrace();
    }
}
```

- Para armazenar/atualizar uma instância basta usar o método *persist* do *EntityManager*.

DAO: recuperando entidade

- Opção 1: usando SQL

```
@NamedQueries( {
    @NamedQuery(name = "Aluno.recuperarUm",
        query = "SELECT d FROM Aluno d WHERE d.matricula = :matricula"),
})

public Aluno getAluno(int nro){
    EntityTransaction tx = manager.getTransaction();
    Aluno aluno = null;
    tx.begin();
    try {
        Query query = manager.createNamedQuery("Aluno.recuperarUm");
        query.setParameter("matricula", nro);
        aluno = (Aluno) query.getSingleResult();
        tx.commit();
    } catch (RuntimeException e) {
        e.printStackTrace();
        tx.rollback();
    }
    return(aluno);
}
```

→ Uma *NamedQuery* deve ser declarada junto da entidade, logo após a anotação *@Entity*

DAO: recuperando entidade

- Opção 2: usando o EntityManager

```
public Aluno recupera(int nro){  
    Aluno aluno = null;  
    try {  
        aluno = manager.find(Aluno.class, nro);  
    } catch (RuntimeException e) {  
        e.printStackTrace();  
    }  
    return(aluno);  
}
```

DAO: remoção de entidade

```
public void remove(int nro){  
    Aluno aluno = null;  
    try {  
        aluno = manager.find(Aluno.class, nro);  
        manager.remove(aluno);  
    } catch (RuntimeException e) {  
        e.printStackTrace();  
    }  
}
```

DAO: recuperando uma lista

```
@NamedQueries( {
    @NamedQuery(name = "Aluno.listarTodos",
        query = "SELECT d FROM Aluno d")})

public List<Aluno> getAllAlunos(){
    EntityTransaction tx = manager.getTransaction();
    List<Aluno> alunos = null;
    tx.begin();
    try {
        Query query = manager.createNamedQuery("Aluno.listarTodos");
        alunos = query.getResultList();
        tx.commit();
    } catch (RuntimeException e) { // <-- importante, agora é runtime
        e.printStackTrace();
        tx.rollback();
    }
    return(alunos);
}
```